Le théorème de Ladner

Floriane Dardard

ENS

21 janvier 2010

Introduction

Il était une fois...la complexité

C'est l'histoire de deux(ou une) classes de langages : P et NP et d'une sous-classe NP-complet. Quelles sont leur relations exactes ?

À quoi ça sert ?

Pour classer un problème dans la classe NP, il est souvent utile et plus simple de montrer qu'il est NP-complet.

On aimerait bien que tous les problèmes soient NP-complets, mais est-ce vraiment le cas ?

Introduction

Il était une fois...la complexité

C'est l'histoire de deux(ou une) classes de langages : P et NP et d'une sous-classe NP-complet. Quelles sont leur relations exactes ?

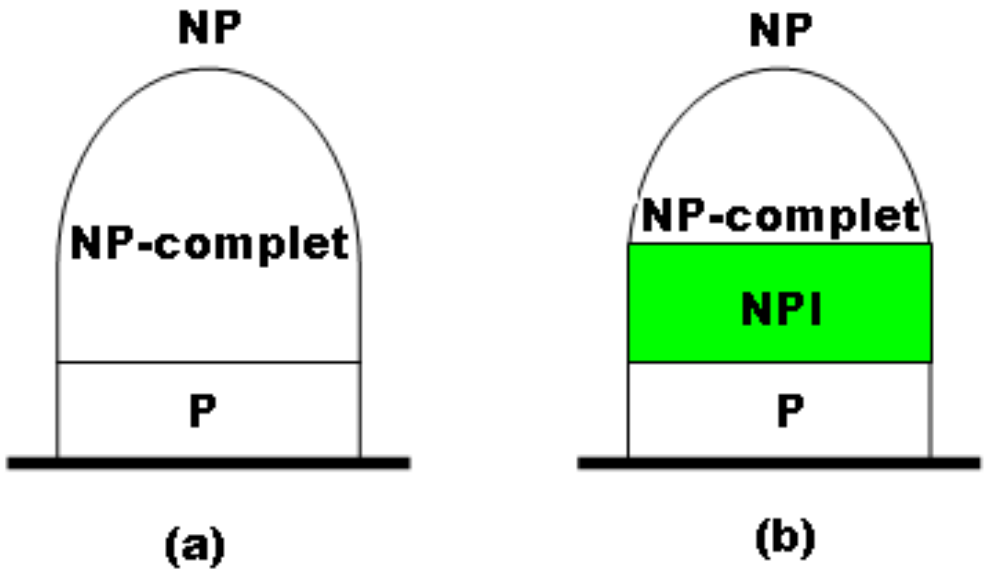
À quoi ça sert ?

Pour classer un problème dans la classe NP, il est souvent utile et plus simple de montrer qu'il est NP-complet.

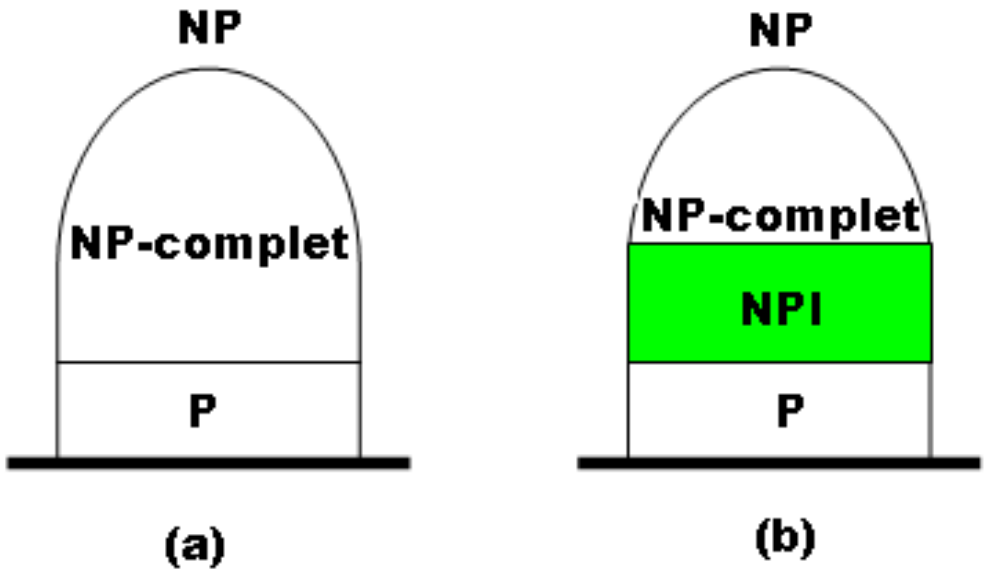
On aimerait bien que tous les problèmes soient NP-complets, mais est-ce vraiment le cas ?

En fait non.

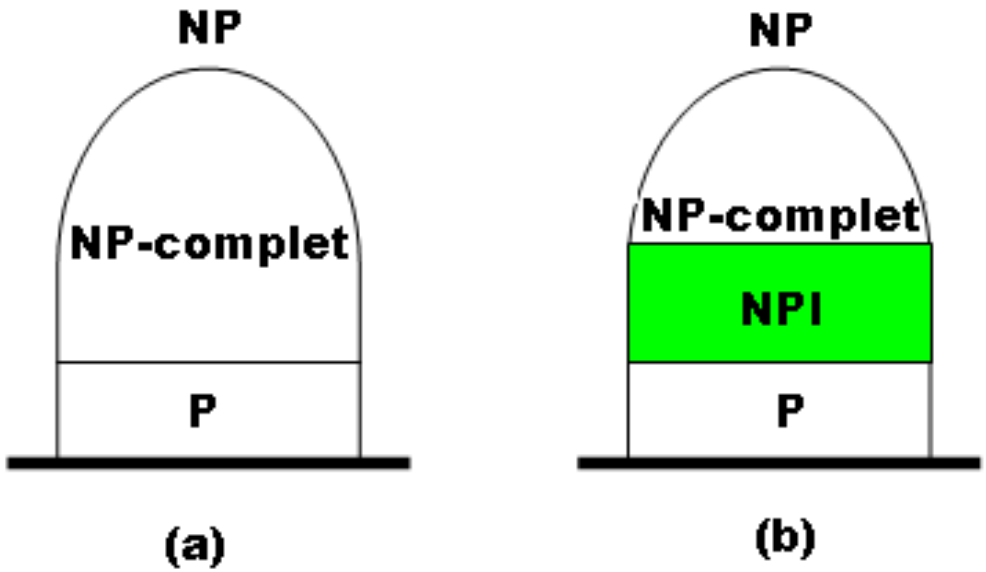
- Le point central de cette question est l'égalité de P et NP.
- En effet, si **P = NP**, alors tout problème dans **NP** est dans **P** et est **NP**-difficile, et alors **P = NP = NP**-complet.
- Par contre, si on suppose **P≠NP**, la question devient : A quoi ressemble la classe de complexité NP ?



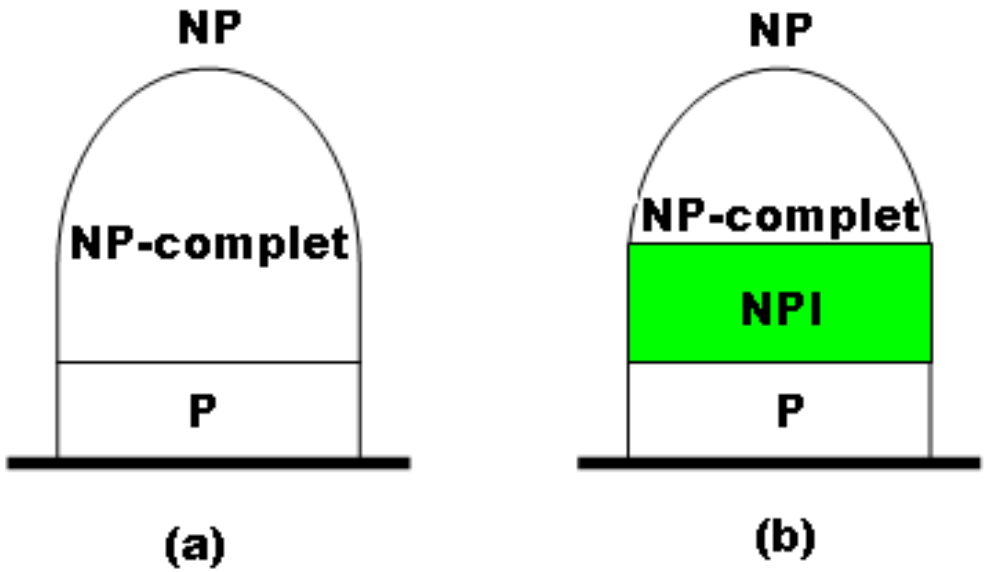
- Le point central de cette question est l'égalité de P et NP.
- En effet, si **P = NP**, alors tout problème dans **NP** est dans **P** et est **NP**-difficile, et alors **P = NP = NP**-complet.
- Par contre, si on suppose **P≠NP**, la question devient : A quoi ressemble la classe de complexité NP ?



- Le point central de cette question est l'égalité de *P* et *NP*.
- En effet, si **P = NP**, alors tout problème dans **NP** est dans **P** et est **NP**-difficile, et alors **P = NP = NP**-complet.
- Par contre, si on suppose **P**$\neq$**NP**, la question devient : A quoi ressemble la classe de complexité NP ?



- Le point central de cette question est l'égalité de *P* et *NP*.
- En effet, si **P = NP**, alors tout problème dans **NP** est dans **P** et est **NP**-difficile, et alors **P = NP = NP**-complet.
- Par contre, si on suppose **P**$\neq$**NP**, la question devient : A quoi ressemble la classe de complexité NP ?



Plan

Le théorème et sa démonstration

- Le théorème
- Démonstration originale

Exemples concrets

- FACTOR
- GRAPHISOMORPHISM

Améliorations du théorème

- Cardinal de NPI
- Densité de NPI

Le théorème

Théorème (Ladner, 1975)

Si **P**$\neq$**NP**, alors il existe un langage dans **NP** qui n'est ni dans **P** ni **NP**-complet.

Une nouvelle classe de langages était née, qu'on appela NPI, c'est-à-dire NP-Intermédiaire.

Plan

Le théorème et sa démonstration

- Le théorème
- Démonstration originale

Exemples concrets

- FACTOR
- GRAPHISOMORPHISM

Améliorations du théorème

- Cardinal de NPI
- Densité de NPI

Démonstration

Le principe de la preuve est, comme souvent en complexité, un argument de diagonalisation.

On va construire un langage L qui est quelquefois SAT (pour la non-appartenance à **P**)et quelquefois le langage vide (pour la non-complétude). On définit le langage A :

$$A = \{\omega / \omega \in \mathbf{SAT} etf(|\omega|) estpair\} \tag{1}$$

avec f une fonction à définir

Démonstration

Le principe de la preuve est, comme souvent en complexité, un argument de diagonalisation.

On va construire un langage L qui est quelquefois SAT (pour la non-appartenance à **P**)et quelquefois le langage vide (pour la non-complétude). On définit le langage A :

$$A = \{\omega / \omega \in \mathbf{SAT} etf(|\omega|) estpair\} \tag{1}$$

avec f une fonction à définir

- On cherche une manière de décrire tous les langages de **P**.
- Soit M_1, M_2,... une énumération des machines de Turing déterministes calculables en un temps polynomial, avec répétitions éventuelles, telles que $M_i(x)$ se calcule en un temps $|x|^i$
- On considère de même une énumération R_i des fonctions calculables en un temps polynomial.
- Pour chaque i, on a deux critères à satisfaire :

$$(2i)A \neq L_{M_i}$$
$$(2i+1)\exists x, x \in \mathbf{SAT}, et R_i(x) \notin A,$$
$$ou x \notin \mathbf{SAT} et R_i(x) \in A$$

- On cherche une manière de décrire tous les langages de **P**.
- Soit M_1, M_2,... une énumération des machines de Turing déterministes calculables en un temps polynomial, avec répétitions éventuelles, telles que $M_i(x)$ se calcule en un temps $|x|^i$
- On considère de même une énumération R_i des fonctions calculables en un temps polynomial.
- Pour chaque i, on a deux critères à satisfaire :

$$(2i)A \neq L_{M_i}$$
$$(2i+1)\exists x, x \in \mathbf{SAT}, et R_i(x) \notin A,$$
$$ou x \notin \mathbf{SAT} et R_i(x) \in A$$

- On cherche une manière de décrire tous les langages de **P**.
- Soit M_1, M_2,... une énumération des machines de Turing déterministes calculables en un temps polynomial, avec répétitions éventuelles, telles que $M_i(x)$ se calcule en un temps $|x|^i$
- On considère de même une énumération R_i des fonctions calculables en un temps polynomial.
- Pour chaque i, on a deux critères à satisfaire :

$$(2i)A \neq L_{M_i}$$

$$(2i+1)\exists x, x \in \mathbf{SAT}, et R_i(x) \notin A,$$

$$ou x \notin \mathbf{SAT} et R_i(x) \in A$$

- On cherche une manière de décrire tous les langages de **P**.
- Soit M_1, M_2,... une énumération des machines de Turing déterministes calculables en un temps polynomial, avec répétitions éventuelles, telles que $M_i(x)$ se calcule en un temps $|x|^i$
- On considère de même une énumération R_i des fonctions calculables en un temps polynomial.
- Pour chaque i, on a deux critères à satisfaire :

$$(2i)A \neq L_{M_i}$$
$$(2i+1)\exists x, x \in \mathbf{SAT}, et R_i(x) \notin A,$$
$$ou x \notin \mathbf{SAT} et R_i(x) \in A$$

On va alors définir f qui donne l'avancée de la satisfaction de tous ces critères. f va croître lentement. On définit : f(0)=f(1)=2 *Intuitivement :*

- A l'étape 2i : On définit

$$f(n) = 2i$$

pour des n de plus en plus grands, jusqu'à ce que le critère $(2i)$ soit satisfait. Si jamais ce critère n'était jamais satisfait pour un certain i, alors on aurait $A = L_{M_i}$ et A serait égal à **SAT** sauf sur un nombre fini d'instances, et cela contredirait le fait que **P**$\neq$**NP**.

- A l'étape 2i+1 : On définit

$$f(n) = 2i + 1$$

jusqu'à ce que le critère $(2i + 1)$ soit satisfait. Si jamais ce critère n'était jamais satisfait pour un certain i, alors A serait fini et **SAT** se réduirait à A par R_i, ce qui impliquerait que **SAT**$\in$**P**, et cela contredirait **P**$\neq$**NP**.

On va alors définir f qui donne l'avancée de la satisfaction de tous ces critères. f va croître lentement. On définit : f(0)=f(1)=2 *Intuitivement :*

- A l'étape 2i : On définit

$$f(n) = 2i$$

pour des n de plus en plus grands, jusqu'à ce que le critère $(2i)$ soit satisfait. Si jamais ce critère n'était jamais satisfait pour un certain i, alors on aurait $A = L_{M_i}$ et A serait égal à **SAT** sauf sur un nombre fini d'instances, et cela contredirait le fait que **P**$\neq$**NP**.

- A l'étape 2i+1 : On définit

$$f(n) = 2i + 1$$

jusqu'à ce que le critère $(2i + 1)$ soit satisfait. Si jamais ce critère n'était jamais satisfait pour un certain i, alors A serait fini et **SAT** se réduirait à A par R_i, ce qui impliquerait que **SAT**$\in$**P**, et cela contredirait **P**$\neq$**NP**.

On va alors définir f qui donne l'avancée de la satisfaction de tous ces critères. f va croître lentement. On définit : f(0)=f(1)=2 *Intuitivement :*

- A l'étape 2i : On définit

$$f(n) = 2i$$

pour des n de plus en plus grands, jusqu'à ce que le critère $(2i)$ soit satisfait. Si jamais ce critère n'était jamais satisfait pour un certain i, alors on aurait $A = L_{M_i}$ et A serait égal à **SAT** sauf sur un nombre fini d'instances, et cela contredirait le fait que **P**$\neq$**NP**.

- A l'étape 2i+1 : On définit

$$f(n) = 2i + 1$$

jusqu'à ce que le critère $(2i + 1)$ soit satisfait. Si jamais ce critère n'était jamais satisfait pour un certain i, alors A serait fini et **SAT** se réduirait à A par R_i, ce qui impliquerait que **SAT**$\in$**P**, et cela contredirait **P**$\neq$**NP**.

Contruction formelle : Soit F la machine de Turing qui calcule f, on doit assurer à la fois la satisfaction des critères déjà mentionnés, et le fait que f va être calculé en $\Theta(n)$.
Pour cela, F va fonctionner en deux étapes distinctes pour le calcul de $f(n)$:

1ère étape : calcul rapide de f
F calcule $f(0), f(1)...$ jusqu'à avoir accompli n opérations. Supposons que la dernière valeur soit $f(i) = k$ (on a évidemment $i \leq n$) Alors on aura f(n)=k ou k+1 selon la deuxième étape.

Contruction formelle : Soit F la machine de Turing qui calcule f, on doit assurer à la fois la satisfaction des critères déjà mentionnés, et le fait que f va être calculé en $\Theta(n)$.
Pour cela, F va fonctionner en deux étapes distinctes pour le calcul de $f(n)$:

1ère étape : calcul rapide de f
F calcule $f(0), f(1)$... jusqu'à avoir accompli n opérations. Supposons que la dernière valeur soit $f(i) = k$ (on a évidemment $i \leq n$) Alors on aura f(n)=k ou k+1 selon la deuxième étape.

2ème étape : satisfaire les critères

- Si $k = 2i$ est pair, alors F calcule $\{M_i(z), SAT(z), F(|z|)\}$ où z est pris dans l'ordre lexicographique sur Σ^* jusqu'à avoir fait n opérations. F cherche un z tel que $A(z) \neq M_i(z)$. Si F trouve un tel z en n opérations, la condition $(2i)$ est satisfaite, et on définit $f(n) = k + 1$. Sinon, $f(n) = k$ et F continue à chercher un z à l'étape d'après.
- Si $k = 2i + 1$ est impair, alors F calcule $\{R_i(z), SAT(z), SAT(R_i(z)), F(|R_i(z)|)\}$ où z est pris dans l'ordre lexicographique sur Σ^* jusqu'à avoir fait n opérations, comme dans le cas précédent. F cherche un z tel que $A(R_i(z)) \neq SAT(z)$ Si F trouve un tel z en n opérations, la condition $(2i)$ est satisfaite, et on définit $f(n) = k + 1$. Sinon, $f(n) = k$.

2ème étape : satisfaire les critères

- Si $k = 2i$ est pair, alors F calcule $\{M_i(z), SAT(z), F(|z|)\}$ où z est pris dans l'ordre lexicographique sur Σ^* jusqu'à avoir fait n opérations. F cherche un z tel que $A(z) \neq M_i(z)$. Si F trouve un tel z en n opérations, la condition $(2i)$ est satisfaite, et on définit $f(n) = k + 1$. Sinon, $f(n) = k$ et F continue à chercher un z à l'étape d'après.
- Si $k = 2i + 1$ est impair, alors F calcule $\{R_i(z), SAT(z), SAT(R_i(z)), F(|R_i(z)|)\}$ où z est pris dans l'ordre lexicographique sur Σ^* jusqu'à avoir fait n opérations, comme dans le cas précédent. F cherche un z tel que $A(R_i(z)) \neq SAT(z)$ Si F trouve un tel z en n opérations, la condition $(2i)$ est satisfaite, et on définit $f(n) = k + 1$. Sinon, $f(n) = k$.

C'est gagné !

On obtient alors F qui calcule bien une fonction entière f en $\Theta(n)$.
De plus, f est croissante et n'est pas ultimement constante, sinon cela contredirait $\mathbf{P} \neq \mathbf{NP}$. Donc les critères sont tous satisfaits, et on a bien construit un langage $A \in \mathbf{NPI}$.

Mais il n'est pas vraiment "naturel" ...

C'est gagné !

On obtient alors F qui calcule bien une fonction entière f en $\Theta(n)$.
De plus, f est croissante et n'est pas ultimement constante, sinon cela contredirait $\mathbf{P} \neq \mathbf{NP}$. Donc les critères sont tous satisfaits, et on a bien construit un langage $A \in \mathbf{NPI}$.

Mais il n'est pas vraiment "naturel"...

Plan

Le théorème et sa démonstration

- Le théorème
- Démonstration originale

Exemples concrets

- FACTOR
- GRAPHISOMORPHISM

Améliorations du théorème

- Cardinal de NPI
- Densité de NPI

FACTOR

- **Entrées :** n entier positif et T une cible
- **Question :** n a-t-il un facteur entre 2 et T (inclus) ?

FACTOR$\in$NP $\cap$ co-NP

- Si jamais **FACTOR** était **NP**-complet, alors ce serait un problème **NP**-complet dans **co-NP**. Cela impliquerait que **NP = co-NP**, ce qui est considéré comme faux (à la majorité informaticienne).
- Si jamais **FACTOR** $\in$ **P**, on pourrait casser RSA facilement...
- Par conséquent, on pense que **FACTOR** n'est ni dans **P** ni **NP**-complet.

FACTOR

- **Entrées :** n entier positif et T une cible
- **Question :** n a-t-il un facteur entre 2 et T (inclus) ?

$$\textbf{FACTOR} \in \textbf{NP} \cap \textbf{co-NP}$$

- Si jamais **FACTOR** était **NP**-complet, alors ce serait un problème **NP**-complet dans **co-NP**. Cela impliquerait que **NP = co-NP**, ce qui est considéré comme faux (à la majorité informaticienne).
- Si jamais **FACTOR** $\in$ **P**, on pourrait casser RSA facilement...
- Par conséquent, on pense que **FACTOR** n'est ni dans **P** ni **NP**-complet.

FACTOR

- **Entrées :** n entier positif et T une cible
- **Question :** n a-t-il un facteur entre 2 et T (inclus) ?

$$\textbf{FACTOR} \in \textbf{NP} \cap \textbf{co-NP}$$

- Si jamais **FACTOR** était **NP**-complet, alors ce serait un problème **NP**-complet dans **co-NP**. Cela impliquerait que **NP = co-NP**, ce qui est considéré comme faux (à la majorité informaticienne).
- Si jamais **FACTOR** $\in$ **P**, on pourrait casser RSA facilement...
- Par conséquent, on pense que **FACTOR** n'est ni dans **P** ni **NP**-complet.

Plan

Le théorème et sa démonstration

- Le théorème
- Démonstration originale

Exemples concrets

- FACTOR
- GRAPHISOMORPHISM

Améliorations du théorème

- Cardinal de NPI
- Densité de NPI

GRAPHISOMORPHISM

- **Entrées :** G et H, deux graphes
- **Question :** G est-il isomorphe à H ?

Savoir s'il est ou non dans P est un problème ouvert... D'autre part, on pense que **GRAPHISOMORPHISM** n'est pas **NP**-complet, car cela ferait s'effondrer de nombreux théorèmes. Néanmoins, on pense que GRAPHISOMORPHISM n'est ni dans **P** ni **NP**-complet.

GRAPHISOMORPHISM

- **Entrées :** G et H, deux graphes
- **Question :** G est-il isomorphe à H ?

Savoir s'il est ou non dans P est un problème ouvert... D'autre part, on pense que **GRAPHISOMORPHISM** n'est pas **NP**-complet, car cela ferait s'effondrer de nombreux théorèmes. Néanmoins, on pense que GRAPHISOMORPHISM n'est ni dans **P** ni **NP**-complet.

Plan

Le théorème et sa démonstration

- Le théorème
- Démonstration originale

Exemples concrets

- FACTOR
- GRAPHISOMORPHISM

Améliorations du théorème

- Cardinal de NPI
- Densité de NPI

Théorème (Ladner)

Si **P**$\neq$**NP**, $\forall L \in$**NP** $\setminus$**P**, $\exists L' \in$**P** $\setminus$**NP** tel que $L' \leq L$, mais $L \not\leq L'$.

Même type de preuve que pour le théorème précédent

Théorème (Ladner)

Si **P**$\neq$**NP**, $\forall L \in$**NP** $\setminus$**P**, $\exists L' \in$**P** $\setminus$**NP** tel que $L' \leq L$, mais $L \not\leq L'$.

Même type de preuve que pour le théorème précédent

En itérant ce théorème, on montre facilement l'existence d'une chaîne infinie L_i de langages dans **NPI** tels que

$$\forall i, L_{i+1} \leq L_i$$
$$L_i \not\leq L_{i+1}$$

En itérant ce théorème, on montre facilement l'existence d'une chaîne infinie L_i de langages dans **NPI** tels que

$$\forall i, L_{i+1} \leq L_i$$
$$L_i \not\leq L_{i+1}$$

Plan

Le théorème et sa démonstration

- Le théorème
- Démonstration originale

Exemples concrets

- FACTOR
- GRAPHISOMORPHISM

Améliorations du théorème

- Cardinal de NPI
- Densité de NPI

Donc la classe **NPI** est infinie. On peut même montrer qu'elle est dense, dans le sens où :
Si A et$B$ sont dans **NPI** tels que

$$A \leq B$$
$$B \not\leq A$$

Alors il existe $C \in$**NPI** un problème tel que

$$A \leq C$$
$$C \not\leq A$$
$$C \leq B$$
$$B \not\leq C$$

Donc la classe **NPI** est infinie. On peut même montrer qu'elle est dense, dans le sens où :
Si A et$B$ sont dans **NPI** tels que

$$A \leq B$$
$$B \not\leq A$$

Alors il existe $C \in$**NPI** un problème tel que

$$A \leq C$$
$$C \not\leq A$$
$$C \leq B$$
$$B \not\leq C$$

Donc la classe **NPI** est infinie. On peut même montrer qu'elle est dense, dans le sens où :
Si A et$B$ sont dans **NPI** tels que

$$A \leq B$$
$$B \not\leq A$$

Alors il existe $C \in$**NPI** un problème tel que

$$A \leq C$$
$$C \not\leq A$$
$$C \leq B$$
$$B \not\leq C$$

Conclusion et questions